ΕΡΓΑΣΙΑ 3η

Ημερομηνία Παράδοσης: 03-02-2021

Φοιτητής: ΓΙΑΝΝΑΚΟΣΙΑΝ ΑΝΕΣΤΗΣ

Τμήμα: ΠΛΣ50-ΗΛΕ45

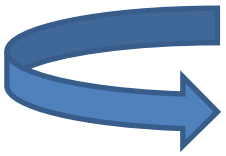
Επιβλέπων Καθηγητής: Κος ΜΑΥΡΟΜΜΑΤΗΣ ΓΕΩΡΓΙΟΣ

Πρόλογος

Πριν ξεκινήσει η επίλυση και επεξήγηση των επιμέρους θεμάτων της Εργασίας 3, να υπενθυμίσω πως το πρόγραμμα για το θέμα 1 δημιουργήθηκε στο **Intellij IDEA** της εταιρείας JETBRAINS, με JDK version 15, αλλά δοκιμάστηκε και στο **Bluej** για την εύρυθμη λειτουργία του. Η εργασία 3 μου φάνηκε πιο απαιτητική, καθώς εισήχθησαν νέα γνωστικά πεδία με τα οποία δεν είχα προηγούμενη επαφή. Παρόλο αυτά προσπάθησα για το καλύτερο δυνατό. Χωρίς να είναι απαραίτητο, ανέβασα μαζί με το Exercise3.zip και τους αλγόριθμους των θεμάτων 3 και 4 Α), υλοποιημένους σε java, αποδεικνύοντας τη λειτουργικότητα τους.

Όσων αφορά την επίλυση του Θέματος 1:

Αποφάσισα να εκμεταλλευτώ τον σκελετό-δομή που δώσατε ως βοήθημα στο .pdf της δεύτερης τηλεσυνάντησης του τμήματος, από την οποία λόγω εργασίας απουσίαζα. Χρησιμοποιώντας βέβαια τις τέσσερις διαφορετικές μεθόδους ***solve()*** στην κλάση ***Sudoku*** για επίλυση του sudoku puzzle, παρατήρησα μεγάλη αύξηση στην επανάληψη κώδικα. Στην προσπάθειά μου να μειώσω λοιπόν την επανάληψη και να κάνω extract τον αλγόριθμο επίλυσης σε ξεχωριστή μέθοδο, παρουσιαζόταν πρόβλημα όταν έπρεπε να καλεστεί διαφορετική δομή. Οπότε το βρήκα ως μία καλή ευκαιρία να δημιουργήσω μία abstract κλάση με τρεις abstract μεθόδους (*signatures*) (.**popPuzzle**(), .**pushPuzzle**(), .**isEmpty**()), τις οποίες χρησιμοποιούν από κοινού οι τέσσερις υλοποιήσεις των δομών stack, queue Με αυτό τον τρόπο έγραψα ένα πιο καθαρό κώδικα χωρίς επανάληψη και δεν είχα πρόβλημα όταν έπρεπε να καλέσω διαφορετική δομή δεδομένων αφού είχε γίνει υλοποίηση της κάθε δομής σε διαφορετική κλάση, η οποία κληρονομούσε από την abstract υπερκλάση.



Θέμα 1: Επίλυση Sudoku με στοίβα και ουρά.

Έστω το γνωστό παζλ Sudoku[1]. Πρόκειται για παζλ όπου ο παίκτης καλείται να τοποθετήσει αριθμούς στα κελιά ενός πίνακα διαστάσεων 9x9, ο οποίος διαμερίζεται σε 9 υποπίνακες διαστάσεων 3x3 έκαστος, υπό τις ακόλουθες προϋποθέσεις:

- Σε κάθε κελί τοποθετείται ένας φυσικός αριθμός από 1 έως 9.
- Κάθε αριθμός πρέπει να εμφανίζεται ακριβώς μια φορά σε κάθε γραμμή του πίνακα, σε κάθε στήλη του πίνακα και σε κάθε υποπίνακα του πίνακα.

Κάθε παζλ Sudoku έχει προσυμπληρωμένες τις τιμές κάποιων κελιών του πίνακα, με τέτοιο τρόπο ώστε το παζλ να έχει ακριβώς μία λύση.

Σκοπός της άσκησης είναι να κατασκευαστεί ένα πρόγραμμα σε Java που να λύνει παζλ Sudoku.

Μπορούμε να λύσουμε ένα παζλ Sudoku χρησιμοποιώντας μια στοίβα ή μια ουρά. Η διαδικασία είναι η εξής:

1. Αρχικά εισάγουμε στη στοίβα ή στην ουρά την αρχική κατάσταση του sudoku. Για την περιγραφή του πίνακα του Sudoku μπορείτε να δημιουργήσετε μια κλάση με όνομα Sudoku, η οποία θα περιέχει ως μεταβλητή κλάσης (μεταξύ άλλων) έναν διδιάστατο πίνακα ακεραίων, διαστάσεων 9x9, καθώς επίσης και όποιες μεθόδους κρίνετε εσείς απαραίτητες. Στον πίνακα θα τοποθετείτε τους αριθμούς από τα αντίστοιχα κελιά του Sudoku, με το 0 να υποδηλώνει το κενό κελί. Μπορείτε επίσης να φτιάξετε μια βοηθητική κλάση, έστω Solver το όνομά της, στην οποία θα βρίσκεται η μέθοδος main και άλλες βοηθητικές μέθοδοι (π.χ., ανάγνωση παζλ από αρχείο, υλοποίηση του αλγορίθμου επίλυσης, κλπ).
2. Στη συνέχεια εκτελούμε επαναληπτικά τα παρακάτω βήματα:
    a. Αφαιρούμε από τη στοίβα ή από την ουρά το πρώτο αντικείμενο.
    b. Για το πρώτο ελεύθερο κελί του Sudoku (με κάποια σειρά, π.χ. από πάνω προς τα κάτω και στη συνέχεια από αριστερά προς τα δεξιά), για κάθε αριθμό που επιτρέπεται να μπει στο κελί αυτό σύμφωνα με τους κανόνες του Sudoku:
        i. Δημιουργούμε ένα νέο αντικείμενο.
        ii. Εάν το αντικείμενο αυτό είναι ένα συμπληρωμένο Sudoku, το παζλ λύθηκε και εμφανίζουμε τον συμπληρωμένο πίνακα στην οθόνη. Το πρόγραμμα τερματίζει.
        iii. Αλλιώς, προσθέτουμε το αντικείμενο στην ουρά ή στην στοίβα.
3. Εάν η ουρά ή η στοίβα αδειάσει χωρίς να βρεθεί λύση, το Sudoku δεν είχε λύση.

Φτιάξτε ένα πρόγραμμα στην Java το οποίο να λύνει παζλ Sudoku. Το πρόγραμμά σας θα διαβάζει από αρχείο την αρχική διάταξη των αριθμών του Sudoku, δηλαδή έναν πίνακα διαστάσεων 9x9. Σε κάθε θέση του πίνακα θα υπάρχει είτε ένας αριθμός από 1 έως 9, ή ο αριθμός 0 αν η θέση αυτή είναι αρχικά κενή. Για διευκόλυνσή, σας δίνονται δέκα έτοιμα αρχεία παζλ Sudoku[1].

Το πρόγραμμά σας θα πρέπει να λύσει παζλ Sudoku με **τέσσερις (4) διαφορετικές δομές δεδομένων**, ειδικότερα χρησιμοποιώντας:

- Δομή δεδομένων 1: Στοίβα υλοποιημένη με την κλάση ArrayList[1] της Java
- Δομή δεδομένων 2: Ουρά υλοποιημένη με την κλάση ArrayList της Java
- Δομή δεδομένων 3: Στοίβα υλοποιημένη με την κλάση Stack[1] της Java
- Δομή δεδομένων 4: Ουρά υλοποιημένη με την κλάση LinkedList[1] της Java

Το πρόγραμμα θα δέχεται ως είσοδο (από τη γραμμή εντολών, μέσω του ορίσματος της μεθόδου main) το όνομα του αρχείου και τον αύξοντα αριθμό της δομής δεδομένων που θα χρησιμοποιηθεί. Υποθέτοντας ότι η κλάση που περιλαμβάνει τη μέθοδο main ονομάζεται Solver, η παρακάτω κλήση:

```
java Solver.class p01.txt 2
```

θα εκτελεί το πρόγραμμά σας με σκοπό να λύσει το παζλ που περιγράφεται στο αρχείο p01.txt, χρησιμοποιώντας ως δομή δεδομένων ουρά υλοποιημένη με την ArrayList.

❖ *Για την επίλυση της άσκησης δημιουργήθηκαν επτά (07) κλάσεις. Σύμφωνα και με την εκφώνηση είναι η public class* ***Sudoku*** *(Sudoku.java) και η public class* ***Solver*** *(Solver.java) στην οποία βρίσκεται η main() μέθοδος και ένα σύνολο SudokuStructures, με τις οι* ***SudokuStructure*** *(SudokuStructure.java),* ***AlsStackStructure*** *(AlsStackStructure.java),* ***AlsQueueStructure*** *(AlsQueueStructure.java),* ***StackStructure*** *(StackStructure.java) και* ***LlQueueStructure*** *(LlQueueStructure.java).*

➢ **Υλοποίηση Solver.java**

Έγινε import των α) ***java.io.FileNotFoundException*** για να επιτευχθεί ο χειρισμός της συγκεκριμένης εξαίρεσης, β) ***java.io.FileReader*** για να χρησιμοποιηθεί η αντίστοιχη κλάση για διάβασμα αρχείου Sudoku, γ) ***java.util.Scanner*** για να χρησιμοποιηθεί η αντίστοιχη κλάση για διάβασμα αρχείου Sudoku.

Α) Μεταβλητές Κλάσης

Οι μία μεταβλητή που δηλώθηκε είναι οι εξής:

- ***private*** *static Scanner* ***sudokufile,*** *στην οποία* θα αποθηκεύσουμε τα περιεχόμενα του αρχικά δοσμένου Sudoku puzzle.

Γ) Ανάλυση μεθόδων

Η κλάση αποτελείται από 3 μεθόδους, ως εξής:

- ***private*** *static int* ***validateArguments****().* Πραγματοποιεί validation στα arguments ***args[0]*** και ***args[1]*** και επιστρέφει ***int choice*** το οποίο γίνεται διαβάζοντας το args[1] και μετατρέποντας το από αλφαριθμητικό σε integer με την εντολή ***Integer.parseInt(args[1])***. Αυτό γίνεται μέσα σε μία try..catch.

- ***public*** *static int[][]* ***readFile****(String filename).* Η συγκεκριμένη μέθοδος διαβάζει με τη χρήση της ***Scanner*** και της ***FileReader*** το αρχείο Sudoku σύμφωνα με τη διεύθυνση του αρχείου που δίνεται ως παράμετρος ***(String filename).*** Αυτό γίνεται μέσα σε μία try..catch. Αν έχει περιεχόμενα το αρχείο, τα διατρέχει και αναθέτει στην ***int[][] sudokupuzzle*** να τα αποθηκεύσει.

- ***public*** *static void* ***main****(String[] args).* Η ***main*** ελέγχει και επικυρώνει(validates) τα arguments που καταχωρούνται στο πρόγραμμα όταν το τρέχουμε. Αναθέτει στην τοπική μεταβλητή *int* ***structure*** αυτό που επιστρέφει η ***validateArguments()***. Διαβάζει το sudoku puzzle από το αρχείο που του υποδεικνύουμε και το αποθηκεύει στην τοπική μεταβλητή *int[][]* ***sudokuPuzzle***. Δημιουργεί νέο αντικείμενο της κλάσης Sudoku με όνομα ***sudoku*** το οποίο παίρνει ως παραμέτρους το αρχικό *sudoku puzzle* και τον αριθμό ***structure*** που καθορίζει ποια μέθοδος επίλυσης θα καλεστεί. Το ρόλο του διαχειριστή της διαδικασίας κλήσεως αναλαμβάνει μίας switch. Τέλος η main καλεί την ***displaySudoku()*** η οποία εμφανίζει στο terminal το puzzle 2 φορές, μία φορά το αρχικό puzzle και μία το τελικά διαμορφωμένο puzzle.

➢ **Υλοποίηση Sudoku.java**

Α) Μεταβλητές Κλάσης

Οι 2 μεταβλητές που δηλώθηκαν είναι οι εξής:

- ***private*** *int[][]* ***puzzle****, στην οποία* θα αποθηκευτούν τα περιεχόμενα του αρχικά δοθέντος Sudoku puzzle, για να χρησιμοποιηθούν κατά τη διάρκεια επίλυσης του. Εκεί θα αποθηκευτεί και το τελικά διαμορφωμένο Sudoku, εφόσον υπάρχει λύση.

- ***private*** *SudokuStructure* ***structure****, η μεταβλητή στην οποία* θα αποθηκευτεί η δομή ανάλογα με τη μέθοδο επίλυσης που ζητήθηκε.

Β) Ανάλυση Κατασκευαστή

Η κλάση αποτελείται από 1 κατασκευαστή, ως εξής:

- ***public Sudoku****(int[][] sudokuPuzzle, SudokuStructure structure)*. Ο κατασκευαστής της κλάσης, παίρνει ως όρισμα το αρχικά δοσμένο Sudoku puzzle και το αναθέτει στην μεταβλητή κλάσης ***puzzle***. Τέλος αναθέτει την μέθοδο επίλυσης στη μεταβλητή κλάσης ***structure*** και προσθέτει το αρχικό puzzle στην κατάλληλη δομή.

Γ) Ανάλυση μεθόδων

Η κλάση αποτελείται από 8 μεθόδους, ως εξής:

- ***public*** *boolean* ***solve()***. Η βασικότερη μέθοδος της κλάσης, η οποία χρησιμοποιεί και διατηρεί αναλλοίωτο τον αλγόριθμο επίλυσης Sudoku, αλλάζοντας μόνο τον τρόπο που γίνεται η προσθήκη, η αφαίρεση και τον έλεγχο αν είναι άδεια η δομή, ανάλογα πάντα με την μέθοδο που ζητείται κατά την εκτέλεση του προγράμματος.

- ***public*** *void* ***displaySudoku()***. Διατρέχει τον πίνακα this.***puzzle*** και τον εκτυπώνει στο terminal.

- ***public*** *boolean* ***hasEmptyCells(****int[][] array****)***. Διατρέχει τον πίνακα που δίνεται ως παράμετρος, αναζητώντας κελιά που να έχουν 0. Αν εντοπίσει κάποιο επιστρέφει true, αν όχι επιστρέφει false.

- ***public*** *int[][]* ***copyArray(****int[][] array****)***. Διατρέχει τον πίνακα που δίνεται ως παράμετρος και αποθηκεύει όλα τα στοιχεία του πίνακα σε έναν καινούριο.

- ***public*** *boolean* ***isInRow(****int row, int column, int[][]puzzle****)***. Διατρέχει τον πίνακα που δίνεται ως παράμετρος και προσπαθεί να εντοπίσει εάν ο πιθανός αριθμός υπάρχει στην γραμμή που εξετάζεται εκείνη τη στιγμή.

- ***public*** *boolean* ***isInColumn(****int row, int column, int[][]puzzle****)***. Διατρέχει τον πίνακα που δίνεται ως παράμετρος και προσπαθεί να εντοπίσει εάν ο πιθανός αριθμός υπάρχει στην στήλη που εξετάζεται εκείνη τη στιγμή.

- ***public*** *boolean* ***isInSubGrid(****int row, int column, int number, int[][]puzzle****)***. Διατρέχει τον πίνακα που δίνεται ως παράμετρος και προσπαθεί να εντοπίσει εάν ο πιθανός αριθμός υπάρχει στον υποπίνακα 3x3 που εξετάζεται εκείνη τη στιγμή.

- ***public*** *boolean* ***followTheRules(****int[][] puzzle, int row, int column, int number****)***. Καλεί τις μεθόδους ***isInRow(), isInColumn(), isInSubGrid().*** Αν περάσει επιτυχώς τον έλεγχο ο πιθανός αριθμός, σημαίνει ότι ικανοποιεί τους κανόνες συμπλήρωσης του sudoku puzzle.

Ανάλυση πακέτου SudokuStructures

Το πακέτο αποτελείται από 5 κλάσεις:

1) ***abstract public class SudokuStructure***

Είναι η υπερκλάση η οποία αποτελείται από 3 signatures κοινά για τις 4 υποκλάσεις της.

- ***abstract public int[][] popPuzzle****(); Είναι υπεύθυνη για την αφαίρεση των στοιχείων των δομών*
- ***abstract public void pushPuzzle****(); Είναι υπεύθυνη για την προσθήκη των στοιχείων των δομών*
- ***abstract public boolean isEmpty****(); Είναι υπεύθυνη να ελέγχει αν είναι άδεια κάποια δομή*

2) ***public class AlsStackStructure extends*** *SudokuStructure*
Κληρονομεί από την υπερκλάση τις μεθόδους της και τις κάνει override. Για την αφαίρεση του τελευταίου στοιχείου, χρησιμοποιεί την .remove(index structure.size() -1). Για την προσθήκη στοιχείων την .add(object) και για τον έλεγχο αν είναι άδεια την .isEmpty().

3) ***public class AlsQueueStructure extends*** *SudokuStructure*
Κληρονομεί από την υπερκλάση τις μεθόδους της και τις κάνει override. Για την αφαίρεση του πρώτου στοιχείου, χρησιμοποιεί την .remove(index 0). Για την προσθήκη στοιχείων την .add(object) και για τον έλεγχο αν είναι άδεια την .isEmpty().

4) ***public class StackStructure extends*** *SudokuStructure*
Κληρονομεί από την υπερκλάση τις μεθόδους της και τις κάνει override. Για την αφαίρεση του τελευταίου στοιχείου, χρησιμοποιεί την .pop(). Για την προσθήκη στοιχείων την .push(object) και για τον έλεγχο αν είναι άδεια την .empty().

5) ***public class LlQueueStructure extends*** *SudokuStructure*
Κληρονομεί από την υπερκλάση τις μεθόδους της και τις κάνει override. Για την αφαίρεση του πρώτου στοιχείου, χρησιμοποιεί την .removeFirst(). Για την προσθήκη στοιχείων την .addLast(object) και για τον έλεγχο αν είναι άδεια την .isEmpty().

Main Algorithm στη μέθοδο .solve() της κλάσης Sudoku

```
public boolean solve() {

    while (!structure.isEmpty()) {
        int[][] currentPuzzle = structure.popPuzzle();

        for (int row = 0; row < currentPuzzle.length; row++) {
            for (int column = 0; column < currentPuzzle.length; column++) {

                if (currentPuzzle[row][column] == 0) {
                    for (int number = 1; number <= currentPuzzle.length; number++) {
                        if (this.followTheRules(currentPuzzle, row, column, number)) {
                            int[][] newPuzzle = this.copyArray(currentPuzzle);
                            newPuzzle[row][column] = number;
                            structure.pushPuzzle(newPuzzle);
                            if (this.solve()) {
                              return true;
                            }
                        }
                    }
                    if (this.hasEmptyCells(currentPuzzle)) {
                      return false;
                    }
                }
            }
        }
        this.puzzle = currentPuzzle;
    }
      return true;
}
```

Θέμα 2: Αλγόριθμοι ταξινόμησης

Για κάθε έναν από τους παρακάτω αλγορίθμους επιδείξτε την ταξινόμηση των στοιχείων του παρακάτω πίνακα 10 διαφορετικών αριθμών, βήμα προς βήμα. Οι θέσεις του πίνακα αριθμούνται από 1 έως 10.

8	3	9	1	0	4	2	7	6	5

Ως βήμα θεωρείστε τις περιπτώσεις που γίνεται αντιμετάθεση (swap) των στοιχείων δύο θέσεων του πίνακα. Τα δύο στοιχεία που αλλάζουν θέση θα φαίνονται με χρωματιστή επισήμανση.

α) **Ταξινόμηση με εισαγωγή**

Στον αλγόριθμο αυτό θα αναφέρετε την τιμή της μεταβλητής j κάθε φορά που γίνεται αντιμετάθεση δύο στοιχείων (δεν χρειάζεται να αναφέρετε την τιμή της μεταβλητής i). Ως υπόδειξη σας δίνεται η πρώτη αντιμετάθεση:

β) **Ταξινόμηση με επιλογή**

Στον αλγόριθμο αυτό θα αναφέρετε την τιμή της μεταβλητής i κάθε φορά που γίνεται αντιμετάθεση δύο στοιχείων (δεν χρειάζεται να αναφέρετε την τιμή της μεταβλητής j). Ως υπόδειξη σας δίνεται η πρώτη αντιμετάθεση:

γ) **Ταξινόμηση φυσαλίδας**

Στον αλγόριθμο αυτό θα αναφέρετε την τιμή της μεταβλητής i κάθε φορά που γίνεται αντιμετάθεση δύο στοιχείων (δεν χρειάζεται να αναφέρετε την τιμή της μεταβλητής j). Ως υπόδειξη σας δίνεται η πρώτη αντιμετάθεση:

δ) **Γρήγορη ταξινόμηση**

Στον αλγόριθμο αυτό θα αναφέρετε τις τιμές των μεταβλητών left, right και pivot για κάθε αναδρομική κλήση της quicksort (συμπεριλαμβανομένης της αρχικής), ακολουθούμενες από όλες τις αντιμεταθέσεις τιμών του πίνακα που έγιναν (δεν χρειάζεται να αναφέρετε τις τιμές των i και j για κάθε αντιμετάθεση, ενώ η τελευταία αντιμετάθεση εννοείται ότι προκύπτει από την γραμμή στην εντολή 1.c.i, εκτός και αν το j έχει γίνει ίσο με left).

ε) **Ταξινόμηση με συγχώνευση**

Ο αλγόριθμος αυτός προϋποθέτει την ύπαρξη ενός βοηθητικού πίνακα B, ίσου (τουλάχιστον) μεγέθους με τον προς ταξινόμηση πίνακα A. Στον βοηθητικό πίνακα γίνεται η συγχώνευση των υποπινάκων του αρχικού πίνακα, πριν κάθε συγχωνευμένος υποπίνακας αντιγραφεί στις αντίστοιχες θέσεις του αρχικού.

Δείξτε όλες τις κλήσεις της MergeSort και της Merge, με τη σειρά που συμβαίνουν και με τα ορίσματά τους, καθώς και τη σταδιακή δημιουργία του πίνακα B για κάθε κλήση της Merge (το κελί B[temp] που λαμβάνει τιμή στη γραμμή 2.a.i ή τη γραμμή 2.a.ii να φαίνεται χρωματισμένο).

α) **Ταξινόμηση με εισαγωγή - InsertionSort (A[] , n)**

8	**3**	**9**	**1**	**0**	**4**	**2**	**7**	**6**	**5**

j=2

3	**8**	9	1	0	4	2	7	6	5

j=3 => no swap

j=4

3	8	**1**	**9**	0	4	2	7	6	5
3	**1**	**8**	9	0	4	2	7	6	5
1	**3**	8	9	0	4	2	7	6	5

j=5

1	3	8	**0**	**9**	4	2	7	6	5
1	3	**0**	**8**	**9**	4	2	7	6	5
1	**0**	**3**	8	9	4	2	7	6	5
0	**1**	3	8	9	4	2	7	6	5

j=6

0	1	3	8	**4**	**9**	2	7	6	5
0	**1**	**3**	**4**	**8**	9	2	7	6	5

j=7

0	1	3	4	8	**2**	**9**	7	6	5
0	1	3	4	**2**	**8**	9	7	6	5
0	1	3	**2**	**4**	8	9	7	6	5
0	1	**2**	**3**	4	8	9	7	6	5

j=8

0	1	2	3	4	8	**7**	**9**	6	5
0	1	2	3	4	**7**	**8**	9	6	5

j=9

0	1	2	3	4	7	8	**6**	**9**	5

0	1	2	3	4	7	**6**	**8**	9	5
0	1	2	3	4	**6**	**7**	8	9	5

j=10

0	1	2	3	4	6	7	8	**5**	**9**
0	1	2	3	4	6	7	**5**	**8**	9
0	1	2	3	4	6	**5**	**7**	**8**	9
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

β) <u>Ταξινόμηση με επιλογή – SelectionSort (A[] , n)</u>

8	**3**	**9**	**1**	**0**	**4**	**2**	**7**	**6**	**5**

i=1 **A** [5] ↔ **A** [1]

0	3	9	1	**8**	4	2	7	6	5

i=2 **A** [4] ↔ **A** [2]

0	**1**	9	**3**	8	4	2	7	6	5

i=3 **A** [7] ↔ **A** [3]

0	1	**2**	3	8	4	**9**	7	6	5

i=4 **no swap**

i=5 **A** [6] ↔ **A** [5]

0	1	2	3	**4**	**8**	9	7	6	5

i=6 **A** [10] ↔ **A** [6]

0	1	2	3	4	**5**	9	7	6	**8**

i=7 **A** [9] ↔ **A** [7]

0	1	2	3	4	5	**6**	7	**9**	8

i=8 **=> no swap**

i=9 **A** [10] ↔ **A** [9]

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

γ) **Ταξινόμηση φυσαλίδας – BubbleSort (int A[] , int n)**

8	**3**	**9**	**1**	**0**	**4**	**2**	**7**	**6**	**5**

i=1

3	**8**	9	1	0	4	2	7	6	5
3	8	**1**	**9**	0	4	2	7	6	5
3	8	1	**0**	**9**	4	2	7	6	5
3	8	1	0	**4**	**9**	2	7	6	5
3	8	1	0	4	**2**	**9**	7	6	5
3	8	1	0	4	2	**7**	**9**	6	5
3	8	1	0	4	2	7	**6**	**9**	5
3	8	1	0	4	2	7	6	**5**	**9**

i=2

3	**1**	**8**	0	4	2	7	6	5	9
3	1	0	**8**	**4**	2	7	6	5	9
3	1	0	**4**	**8**	2	7	6	5	9
3	1	0	4	**2**	**8**	7	6	5	9
3	1	0	4	2	**7**	**8**	6	5	9
3	1	0	4	2	7	**6**	**8**	5	9
3	1	0	4	2	7	6	**5**	**8**	9

i=3

1	**3**	0	4	2	7	6	5	8	9
1	**0**	**3**	4	2	7	6	5	8	9
1	0	3	**2**	**4**	7	6	5	8	9
1	0	3	2	4	**6**	**7**	5	8	9
1	0	3	2	4	6	**5**	**7**	8	9

i=4

0	**1**	3	2	4	6	5	7	8	9
0	1	**2**	**3**	4	6	5	7	8	9
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

δ) **Γρήγορη ταξινόμηση – QuickSort (1, n, A)**

8	**3**	**9**	**1**	**0**	**4**	**2**	**7**	**6**	**5**

left=1, right=10, pivot=8

8	3	**5**	1	0	4	2	7	6	**9**
6	3	5	1	0	4	2	7	**8**	9

left=1, right=8, pivot=6

2	3	5	1	0	4	**6**	7	8	9

left=1, right=6, pivot=2

2	**0**	5	1	**3**	4	6	7	8	9
2	0	**1**	**5**	3	4	6	7	8	9
1	0	**2**	5	3	4	6	7	8	9

left=1, right=2, pivot=1

0	**1**	2	5	3	4	6	7	8	9

left=4, right=6, pivot=5

0	1	2	**4**	3	**5**	6	7	8	9

left=4, right=5, pivot=4

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

ε) **Ταξινόμηση με συγχώνευση – MergeSort (1, n, A, B)**

8	3	9	1	0	4	2	7	6	5

MergeSort (1,10)

MergeSort (1,5)

MergeSort (1,3)

MergeSort (1,2)

MergeSort (1,1)

MergeSort (2,2)

MergeSort (1,1,2)

3	**8**								

MergeSort (1,10)

MergeSort (1,5)

MergeSort (1,3)

MergeSort (3,3)

MergeSort (1,2,3)

3	8	**9**							

MergeSort (1,10)

MergeSort (1,5)

MergeSort (4,5)

MergeSort (4,4)

MergeSort (5,5)

MergeSort (4,4,5)

3	8	9	**0**	**1**					

MergeSort (1,10)

MergeSort (1,5)

MergeSort (1,3,5)

MergeSort (1,10)

MergeSort (6,10)

MergeSort (6,8)

MergeSort (6,7)

MergeSort (6,6)

MergeSort (7,7)

MergeSort (6,6,7)

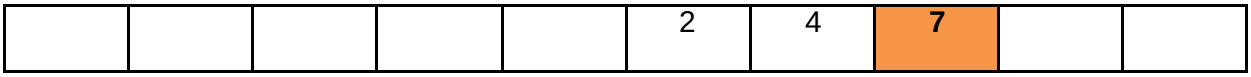
MergeSort (1,10)

MergeSort (6,10)

MergeSort (6,8)

MergeSort (8,8)

MergeSort (6,7,8)



MergeSort (1,10)

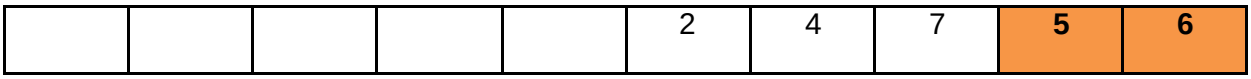
MergeSort (6,10)

MergeSort (9,10)

MergeSort (9,9)

MergeSort (10,10)

MergeSort (9,9,10)



MergeSort (1,10)

MergeSort (6,10)

MergeSort (6,8,10)

					2	4	5	6	7

MergeSort (1,10)

MergeSort (1,5,10)

0	1	2	3	4	5	6	7	8	9

Θέμα 3: Σχεδίαση αλγορίθμου αναζήτησης: Εύρεση μεσαίου στοιχείου πίνακα

Προτείνετε έναν αλγόριθμο που να βρίσκει τον μεσαίο (median) αριθμό σε έναν μη-ταξινομημένο πίνακα A με n διαφορετικούς ακεραίους αριθμούς (θετικούς και αρνητικούς), όπου το n είναι περιττός αριθμός (οπότε ο μεσαίος αριθμός είναι μοναδικός), χωρίς να επιτρέπεται να αλλάξετε τη θέση των αριθμών του A (π.χ., δεν επιτρέπεται να ταξινομήσετε τον A) και χωρίς να έχετε πρόσβαση σε επιπλέον μνήμη πέραν όσης θα χρειαστείτε για τις τοπικές μεταβλητές του αλγορίθμου (π.χ., δεν μπορείτε να δεσμεύσετε μνήμη ίση με το μέγεθος του πίνακά σας, ώστε να προβείτε σε ταξινόμηση, μπορείτε ωστόσο να χρησιμοποιήσετε όση μνήμη θέλετε, το μέγεθος της οποίας όμως θα πρέπει να είναι σταθερό και ανεξάρτητο από το μέγεθος του πίνακα A). Προσπαθήστε να τον βελτιστοποιήσετε όσο περισσότερο γίνεται.

Ποιες είναι οι πολυπλοκότητες O και Ω του αλγορίθμου σας;

Έστω A[i,i+1,…,n-1,n] μη-ταξινομημένος (unsorted) πίνακας n μοναδικών (unique) ακεραίων αριθμών, όπου n=περιττός αριθμός. Από αυτό συμπεραίνουμε ότι το μεσαίο στοιχείο (median) του πίνακα εάν ο πίνακας ήτο ταξινομημένος θα βρισκόταν στη θέση (n+1)/ 2. Επίσης θεωρούμε ως index του πρώτου στοιχείου του πίνακα τη θέση i=1, array[1].

Algorithm:

```
(int A[], int n)
medianPosition= n+1/2 ;
median = A [medianPosition] ;
int greater; //όσα στοιχεία είναι μεγαλύτερα από αυτό που ελέγχεται κάθε φορά
int less;  //όσα στοιχεία είναι μικρότερα από αυτό που ελέγχεται κάθε φορά

1.  for ( i=1 ; i ≤ n ; i++ ) {
        greater = 0 ;
        less = 0 ;
          2.  for ( j=1 ; j ≤ n ; j++ ) {

                    if   j = i
                       continue ;
                    if  A[j] > A[i]
                       greater++ ;
                    else   less++ ;
              } end for
                    If  greater = less
                        median = A[i];
    } end for
```

Πολυπλοκότητα αλγορίθμου worst case → **O(n^2).** Αυτό απορρέει από την περίπτωση που το μεσαίο στοιχείο του πίνακα θα βρίσκεται στην τελευταία θέση του πίνακα, άρα θα τρέξουν και οι δύο for…loops μέχρι το n.

Πολυπλοκότητα αλγορίθμου best case → **Ω(n).** Αυτό απορρέει από την περίπτωση που το μεσαίο στοιχείο του πίνακα θα βρίσκεται στην αρχική θέση του πίνακα, άρα η πρώτη for…loop θα κάνει μόνο ένα run για i=1 και η δεύτερη θα τρέξει μέχρι και j=n, για να γίνει τελικά η σύγκριση μεγαλύτερων και μικρότερων στοιχείων, σε σχέση με το πρώτο στοιχείο στο index array[1].

Θέμα 4: Πολυπλοκότητα αλγορίθμων

Α) Η πράξη της *συνέλιξης* (*convolution*) στα συνελικτικά νευρωνικά δίκτυα (convolutional neural networks) λειτουργεί ως εξής: Έχουμε έναν μεγάλο πίνακα εισόδου, έστω τετράγωνο διδιάστατο, διαστάσεων $N \times N$, και έναν σημαντικά μικρότερο πίνακα διαστάσεων $K \times K$ ($K << N$). Έστω **Α** ο πρώτος πίνακας και **Β** ο δεύτερος. Ο πίνακας **Β** αποκαλείται συνήθως «*φίλτρο*».

Το φίλτρο **Β** εφαρμόζεται σε κάθε υποπεριοχή του πίνακα **Α** διαστάσεων $K \times K$ ως εξής: Τα στοιχεία του φίλτρου πολλαπλασιάζονται ένα-προς-ένα με τα αντίστοιχα στοιχεία της υποπεριοχής του πίνακα **Α** και τα επιμέρους γινόμενα αθροίζονται. Το αποτέλεσμα τοποθετείται σε μια θέση του πίνακα αποτελέσματος **Γ**, ο οποίος έχει διαστάσεις $(N - K + 1) \times (N - K + 1)$.

Στο παρακάτω παράδειγμα φαίνονται ο πίνακας εισόδου **Α** διαστάσεων 5×5, το φίλτρο **Β** διαστάσεων 2×2 και το αποτέλεσμα, ο πίνακας **Γ**, διαστάσεων 4×4:

6	27	14	8	19
30	19	29	5	4
5	4	2	27	20
9	12	26	11	1
10	25	25	8	29

Α

-0,6	-0,9
0,7	0,9

Β

10,2	10,6	9,2	-14,8
-28	-32,9	3,8	30,3
10,5	27,6	2,6	-25,6
13,3	9,4	-0,8	24,2

Γ

Η κίτρινη περιοχή του πίνακα **Α**, σε συνδυασμό με ολόκληρο τον πίνακα **Β** έδωσαν ως αποτέλεσμα το πράσινο κελί του πίνακα **Γ**. Ειδικότερα, ισχύει:

$$4 \times -0.6 + 2 \times -0.9 + 12 \times 0.7 + 26 \times 0.9 = 27.6$$

Δοθέντων δύο διδιάστατων πινάκων **Α** και **Β**, διαστάσεων $N \times N$ και $K \times K$ αντίστοιχα, γράψτε τον ψευδοκώδικα υπολογισμού του πίνακα **Γ**. Ποια η πολυπλοκότητα υπολογισμού του;

Β)

i) Βρείτε το άνω φράγμα (μεγάλο Ο) πολυπλοκότητας των παρακάτω αναδρομικών συναρτήσεων, χρησιμοποιώντας τη μέθοδο της αντικατάστασης και ισχυρής επαγωγής:

α) $T(n) = T(n-3) + n^3$

Υπόδειξη: Δοκιμάστε διάφορες δυνάμεις του n (π.χ., n^2, n^3, κλπ)

β) $T(n) = T(n-1) + 3T(n-2) + 3T(n-3)$

Υπόδειξη: Δοκιμάστε διάφορες εκθετικές συναρτήσεις του n (π.χ., 2^n, 3^n, κλπ)

ii) Βρείτε τις πολυπλοκότητες των παρακάτω αναδρομικών συναρτήσεων χρησιμοποιώντας το κεντρικό θεώρημα:

α) $T(n) = 4T(n/2) + n/logn$

β) $T(n) = 6T(n/3) + n^2 logn$

Θέμα 4

Α) Θα προσπαθήσω να αναλύσω τη λογική πίσω από τον αλγόριθμο που παραθέτω κάτωθεν. Χρησιμοποίησα δύο for…loops να τρέξω το σύνολο των κελιών του πίνακα Α σύμφωνα με την πράξη συνέλιξης N - K + 1 ώστε να τοποθετείται κάθε φορά το αποτέλεσμα της πράξεων στο κατάλληλο κελί του πίνακα Γ. Επίσης θεωρώντας την ύπαρξη του πίνακα Β(φίλτρο) μεγίστης σημασίας, είδα την ανάγκη διάτρεξης του αντίστοιχα από 2 for...loop, ανεβάζοντας παράλληλα και την πολυπλοκότητα. Μελετώντας και το παράδειγμα της 2ης τηλεσυνάντησης του τμήματος, οδηγήθηκα σε αυτήν την απόφαση εισαγάγοντας συνολικά 4 μεταβλητές i,j,k,d.

Η πρώτη μου απόπειρα επίλυσης του προβλήματος υλοποιήθηκε από 3 for...loops και μια ενιαία πράξη σύμφωνα με τη δοθείσα της εκφώνησης, αλλά ένιωσα ότι δεν είναι η καταλληλότερη απάντηση για κάθε πιθανό n και k των τετραγωνικών πινάκων Α και Β αντίστοιχα. Επομένως έβαλα ως κεντρική πράξη, μέσα στην τελευταία for..loop, την ***C[i][j] += A[i+k][j+d] * B[k][d];***, η οποία αποτελεί και μια πιο γενικευμένη προσέγγιση ώστε ανάλογα με τα μεγέθη N και K να εκτελεί και τις αντίστοιχες προσθέσεις. Αυτό το έκανα για να πιάσω και περιπτώσεις πέραν της περίπτωσης που περιγράφει η εκφώνηση και να αποφύγω περιγραφή index 2d πίνακα, τύπου k-1, k+1, i+1, j+1.

Παρόλα αυτά ανυπομονώ να δω εάν κατάφερα να πλησιάσω το ζητούμενο της άσκησης. Επίσης να τονίσω ότι ως index του στοιχείου στην πρώτη θέση των πινάκων έβαλα το 0, προσομοιάζοντας την υλοποίηση στη java.

Algorithm:

(int A[][] (N*N), int B[][] (K*K), int C[][], (N-K+1)*(N-K+1)

C.length = [A.length - B.length + 1][A.length - B.length + 1]

```
1. for (int i = 0; i < C.length; i++) {
    2. for (int j = 0; j < C.length; j++) {
            i. C[i][j] = 0;
        3. for (int k = 0; k < B.length; k++) {
            4. for (int d = 0; d< B.length; d++){
                i. C[i][j] += A[i+k][j+d] * B[k][d];
            } end for
        } end for
    } end for
} end for
```

Η πολυπλοκότητα του αλγορίθμου είναι **O**(**(N-K+1)^2*K^2**). Όταν το K είναι κατά πολύ μικρότερο του N (K <<<< N) ουσιαστικά δεν επηρεάζει σε σημαντικό βαθμό, ώστε να μπορούμε να θεωρήσουμε πολυπλοκότητα **O**(**(N-K+1)^2**)**.**

Θέμα 4

Β) i) ***α) $T(n) = T(n-3) + n^3$***

β) $T(n) = T(n-1) + 3T(n-2) + 3T(n-3)$

α) ***$T(n) = T(n-3) + n^3$,***

Η εκτίμηση μας είναι ότι το μεγάλο φράγμα ισούται με $O(n^3)$. Για να είναι σωστή η εκτίμηση μας πρέπει να ισχύει $T(n) \leq cn^3$, για κάποια σταθερά $c>0$

- ***Σύμφωνα τη μέθοδο αντικατάστασης:***

$$T(n) = T(n-3) + n^3 = c(n-3)^3$$
$$= cn^3 - 9cn^2 + 27cn - 27c + n^3 =$$
$$= cn^3 - 9cn(n - 3 - n^2/9) - 27c$$
$$\leq cn^3 \text{, το οποίο ισχύει για } c>0,\ 0<n<2$$

άρα $\boxed{T(n) = \Theta(n^3)}$

β) ***$T(n) = T(n-1) + 3T(n-2) + 3T(n-3)$,***

Η εκτίμηση μας είναι ότι το άνω φράγμα ισούται με $O(3^n)$. Για να είναι σωστή η εκτίμηση μας πρέπει να ισχύει $T(n) \leq c3^n$, για κάποια σταθερά $c>0$

- ***Σύμφωνα τη μέθοδο αντικατάστασης:***

$$T(n) = T(n-1) + 3T(n-2) + 3T(n-3)$$
$$= c3^{n-1} + 3c3^{n-2} + 3c3^{n-3}$$
$$= c3^n/3^1 + 3c3^n/3^2 + 3c3^n/3^3$$
$$= (c/3)3^n + (3c/9)3^n + (3c/27)3^n$$
$$= c3^n\ (1/3 + 3/9 + 3/27)$$
$$= c3n\ (3/9 + 3/9 + 1/9)$$
$$= 7/9c3^n$$
$$\leq c3^n \text{ , το οποίο ισχύει για } c>0,\ n>0$$

άρα $\boxed{T(n) = O(3^n)}$

Ύστερα με την ίδια μέθοδο έγινε υπολογισμός και για άνω φράγμα $O(2^n)$ αλλά η εκτίμηση 2^n δεν ικανοποιεί την ***$T(n) \leq c2^n$*** **.**

$$T(n) = c2n-1 + 3c2n-2 + 3c2n-3 = \ldots = 13/8\ c2n \geq c2n \text{ , για κάθε } c>0, n>0$$

ii) ***α) $T(n) = 4T(n/2) + n/\log n$***

β) $T(n) = 6T(n/3) + n^2 \log n$

α) ***$T(n) = 4T(n/2) + n/\log n \rightarrow$ αναδρομική εξίσωση της μορφής $T(n) = aT(n/b) + f(n)$,***

- ***<u>Σύμφωνα με το κεντρικό θεώρημα:</u>***

  $a = 4$, $b = 2$, $f(n) = n/\log n$ και $\log_b a = \log_2 4 = 2$

- ***<u>Εξετάζουμε την πρώτη περίπτωση του κεντρικού θεωρήματος:</u>***

Case 1. $f(n) = O(n^{\log_b a - \varepsilon})$, για κάποια σταθερά $\varepsilon > 0$, Τότε $T(n) = \Theta(n^{\log_b a})$

$f(n) = O(n^{\log_b a - \varepsilon}) = O(n^{\log_2 4 - \varepsilon}) = O(n^{2-\varepsilon})$, το οποίο ισχύει

άρα $\boxed{T(n) = \Theta(n^2)}$

β) ***$T(n) = 6T(n/3) + n^2 \log n \rightarrow$ αναδρομική εξίσωση της μορφής $T(n) = aT(n/b) + f(n)$,***

- ***<u>Σύμφωνα με το κεντρικό θεώρημα:</u>***

  $a = 6$, $b = 3$, $f(n) = n^2 \log n$ και $\log_b a = \log_3 6 \approx 1{,}63$

- ***<u>Εξετάζουμε την τρίτη περίπτωση του κεντρικού θεωρήματος:</u>***

Case 3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$, για κάποια σταθερά $\varepsilon > 0$ & $Af(n/b) \le c\, f(n)$, για $c < 1$ $\Rightarrow$ $T(n) = \Theta(f(n))$

$f(n) = O(n^{\log_b a + \varepsilon}) = O(n^{\log_3 6 + \varepsilon}) = O(n^{1,63+\varepsilon})$, το οποίο ισχύει

- ***Άρα να επαληθεύσουμε ότι $6f(n/3) \le c\, n^2 \log n$***

$6(n/3)^2 \log(n/3) = 6n^2/9 \log n/3 = \boxed{2/3\, n^2 \log n/3 \le c\, n^2 \log n}$, το οποίο ισχύει π.χ για $c = 2/3$

άρα $\boxed{T(n) = \Theta(f(n)) = \Theta(n^2 \log n)}$